The Stanford `libcs106` library, Academic Year 2024-25

```
#include "set.h"
```

`class Set<ValueType>`

This class stores a collection of distinct elements.

The set uses a binary search tree (BST) structure internally. Because of this choice of internal representation, the **`ValueType`** for the type of elements stored in a **`Set`** must define a natural ordering through a **`less`** function and/or **`<`** operator so that the elements can be compared and ordered. The range-based for loop will iterate over the elements in sorted order. The Set operations to add/find/remove an element run in O(logN) time.

Constructor

`Set()`	O(1)	Initializes a new set of the specified element type.

Methods

`add(value)`	O(log N)	Adds an element to this set, if it was not already there.
`clear()`	O(N)	Removes all elements from this set.
`contains(value)`	O(log N)	Returns **`true`** if the specified value is in this set.
`difference(otherSet)`	O(N log N)	Subtracts **`otherSet`** from this set. The difference removes those elements that appear in **`otherSet`**.
`equals(set)`	O(N)	Returns **`true`** if the two sets contain the same elements.
`first()`	O(1)	Returns the first value in this set when considered in sorted order.
`intersect(otherSet)`	O(N log N)	Intersects **`otherSet`** with this set. The intersection retains only those elements also contained in **`otherSet`**.
`isEmpty()`	O(1)	Returns **`true`** if this set contains no elements.

isSubsetOf(*otherSet*)	O(N)	Returns **true** if this set is a subset of **otherSet**.
isSupersetOf(*otherSet*)	O(N)	Returns **true** if this set is a superset of **otherSet**.
last()	O(log N)	Returns the last value in this set when considered in sorted order.
mapAll(*fn*)	O(N)	Calls **fn(value)** for each element of this set.
remove(*value*)	O(log N)	Removes an element from this set.
size()	O(1)	Returns the number of elements in this set.
toString()	O(N)	Returns a printable string representation of this set.
unionWith(*otherSet*)	O(N log N)	Unions **otherSet** with this set. The union adds all elements from **otherSet**.

Operators

for (ValueType elem : set)	O(N)	Iterates through the elements in a set.
set1 == set2	O(N)	Evaluates to **true** if **set1** and **set2** contain the same elements.
set1 != set2	O(N)	Evaluates to **true** if **set1** and **set2** are different.
set + value	O(N)	Creates a new set which is the union of **set** with the single **value**.
set1 + set2	O(N log N)	Creates a new set which is the union of **set1** with **set2**.
set += value	O(log N)	Adds the single **value** to **set**.
set1 += set2	O(N log N)	Adds all of the elements from **set2** to **set1**.
set - value	O(N)	Creates a new set which contains all values in **set** minus **value**.
set1 - set2	O(N log N)	Creates a new set containing the elements in **set1** that aren't in **set2**.
set -= value	O(log N)	Removes the single **value** from **set**.
set1 -= set2	O(N log N)	Removes the elements of **set2** from **set1**.
set1 * set2	O(N log N)	Creates a new set which is the intersection of **set1** and **set2**.
set1 *= set2	O(N log N)	Removes any elements from **set1** that are not present in **set2**.

ostream << set	O(N)	Outputs the contents of **set** to the given output stream.
istream >> set	O(N log N)	Reads the contents of the given input stream into **set**.

Constructor detail

Set();

Creates an empty set of the specified element type. You may also optionally provide an initializer list of values. The newly created set will contain those values.

Usage:

```
Set<ValueType> set;
Set<ValueType> set = { value1, value2, value3 };
```

Method detail

---

void add(const ValueType& value);

Adds an element to this set, if it was not already there.

Usage:

```
set.add(value);
```

---

void clear();

Removes all elements from this set.

Usage:

```
set.clear();
```

```
bool contains(const ValueType&  value) const;
```

Returns **true** if the specified value is in this set.

Usage:

```
if (set.contains(value)) ...
```

```
Set& difference(const Set& set2);
```

Removes from this set all elements that are present in **set2**. Returns a reference to this set, which has been modified in place. If you want a new set, consider **set1 - set2** instead.

Usage:

```
set1.difference(set2);
```

```
bool equals(const Set& set2) const;
```

Returns **true** if the two sets contain exactly the same element values. Identical in behavior to the **==** operator.

Usage:

```
if (set1.equals(set2)) ...
```

```
ValueType first() const;
```

Returns the first value in the set when considered in sorted order. If set is empty, **first** signals an error.

Usage:

```
ValueType value = set.first();
```

```
Set& intersect(const Set& set2);
```

Removes from this set all elements that are not present in **set2**. Returns a reference to this set, which has been modified in place. If you want a new set, consider **set1 * set2** instead.

Usage:

```
set.intersect(set2);
```

```
bool isEmpty() const;
```

Returns **true** if this set contains no elements.

Usage:

```
if (set.isEmpty()) ...
```

```
bool isSubsetOf(const Set& set2) const;
```

Returns **true** if every element of this set is contained in **set2**.

Usage:

```
if (set1.isSubsetOf(set2)) ...
```

```
bool isSupersetOf(const Set& set2) const;
```

Returns **true** if every element of **set2** is contained in this set.

Usage:

```
if (set1.isSupersetOf(set2)) ...
```

```
ValueType last() const;
```

Returns the last value in the set when considered in sorted order. If set is empty, **last** signals an error.

Usage:

```
ValueType value = set.last();
```

```
void mapAll(std::function<void (const ValueType&)> fn) const;
```

Iterates through the elements of the set and calls **fn(value)** for each one. The elements are processed in sorted order.

Usage:

```
set.mapAll(fn);
```

```
void remove(const ValueType&  value);
```

Removes an element from this set. If the value was not contained in the set, there is no error and the set remains unchanged.

Usage:

```
set.remove(value);
```

```
int size() const;
```

Returns the number of elements in this set.

Usage:

```
int count = set.size();
```

```
string toString() const;
```

Returns a printable string representation of this set, such as **"{value1, value2, value3}"**. The values are listed in sorted order.

Usage:

```
string str = set.toString();
```

Set& unionWith(const Set& set2);

Adds to this set all elements from **set2**. Returns a reference to this set, which has been modified in place. If you want a new set, consider **set1 + set2** instead.

Usage:

```
set1.unionWith(set2);
```

Operator detail

---

for (ValueType elem : set)

The range-based for loop can be used to iterate through the elements in a collection. The elements in a set are accessed in sorted order. An error is signaled if you attempt to add/remove elements from a collection while iterating over it.

Usage:

```
for (ValueType elem : set) {
        cout << elem << endl;
}
```

---

bool operator==(const Set& set2) const;

Returns **true** if **set1** and **set2** contain the same elements.

Usage:

```
set1 == set2
```

```
bool operator!=(const Set& set2) const;
```

Returns **true** if **set1** and **set2** are different.

Usage:

```
set1 != set2
```

```
Set operator+(const Set& set2) const;
Set operator+(const ValueType& element) const;
```

Creates a new set which is the union of **set1** with **set2** (or with the single **value**).

Usage:

```
set1 + set2
set1 + value
```

```
Set operator*(const Set& set2) const;
```

Creates a new set which is the intersection of **set1** and **set2**.

Usage:

```
set1 * set2
```

```
Set operator-(const Set& set2) const;
Set operator-(const ValueType& value) const;
```

Creates a new set which is the difference of **set1** minus **set2** (or minus the single **value**).

Usage:

```
set1 - set2
set1 - value
```

```
Set& operator+=(const Set& set2);
Set& operator+=(const ValueType& value);
```

Adds all of the elements from **set2** (or the single **value**) to **set1**.

Usage:

```
set1 += set2;
set1 += value;
```

```
Set& operator*=(const Set& set2);
```

Removes any elements from **set1** that are not present in **set2**.

Usage:

```
set1 *= set2;
```

```
Set& operator-=(const Set& set2);
Set& operator-=(const ValueType& value);
```

Removes the all elements in **set2** (or the single **value**) from **set1**.

Usage:

```
set1 -= set2;
```

```
set1 -= value;
```

---

ostream& operator<<(const Set& set);

Outputs the contents of **set** to the given output stream. The output is in the form **{value1, value2, value3}** where elements are listed in sorted order.

Usage:

```
cout << set << endl;
```

---

istream& operator>>(Set& set);

Reads the contents of the given input stream into **set**. Any previous contents of the set are replaced. The input is expected to be in the form **{value1, value2, value3}**. If unable to read a proper set from the stream, the operation results in a stream fail state.

Usage:

```
if (infile >> set) ...
```
